

MODULE1

Module-I

Module Title: Introductory concepts of OOP

Programming Paradigms , Preface to Object-Oriented Programming, Key Concepts OOP- Objects, Classes ,Data Abstraction, Encapsulation, Inheritance, Polymorphism ,Dynamic Binding, Message Passing. Advantages of OOP.

C++ Basics: Parts of C++ program ,Classes in C++, Declaring Objects , public , private and protected keywords , Defining Member Functions, Characteristics of Member Functions ,Array of Objects, Objects as Function Argument

Introduction to OOP

Languages such as c, structured programming became very popular and was the main technique of the 1980's. Structured programming was a powerful tool that enabled programmers to write moderately complex programs fairly easily. However, as the programs grew larger, even the structured approach failed to show the desired result in terms of bug-free, easy-to- maintain, and reusable programs.

Object Oriented Programming (OOP) is an approach to program organization and development that attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several powerful new concepts. It is a new way of organizing and developing programs and has nothing to do with any particular language. However, not all languages are suitable to implement the OOP concepts easily.

Procedural Vs. Object Oriented Programming

1.Procedure-Oriented Programming

A typical structure for procedural programming is shown in fig.1.2. The technique of hierarchical decomposition has been used to specify the tasks to be completed for solving a problem.

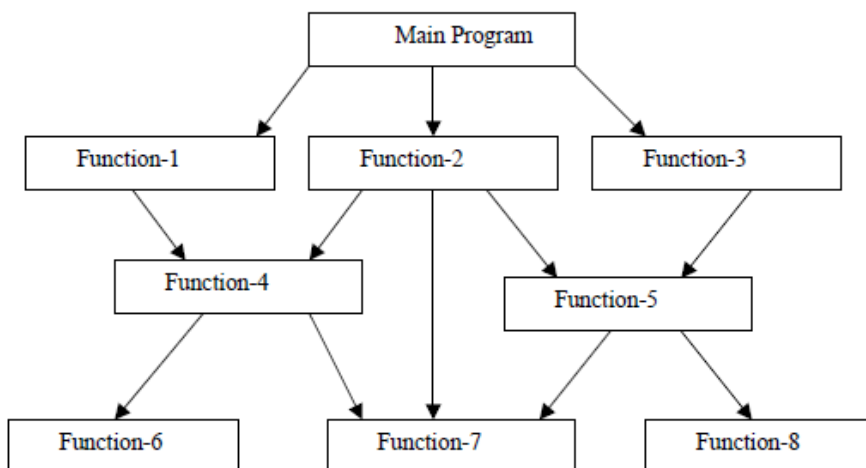


Fig. 1.2 Typical structure of procedural oriented programs

Procedure oriented programming basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as *functions*. We normally use flowcharts to organize these actions and represent the flow of control from one action to another. Each function may have its own local data.

In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides an opportunity for bugs to creep in.

Another serious drawback with the procedural approach is that we do not model real world problems very well. This is because functions are action-oriented and do not really corresponding to the element of the problem.

Some Characteristics exhibited by procedure-oriented programming are:

Emphasis is on doing things (algorithms).

Large programs are divided into smaller programs known as functions.

Most of the functions share global data.

Data move openly around the system from function to function.

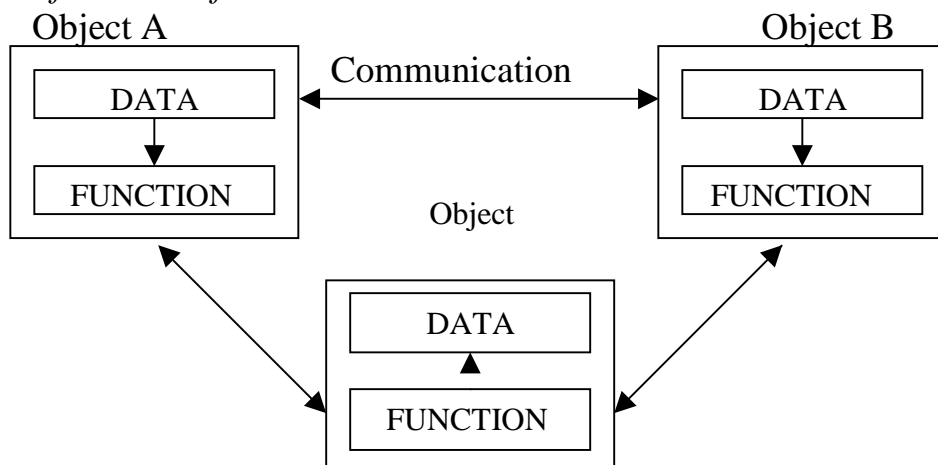
Functions transform data from one form to another.

Employs top-down approach in program design.

2.Object Oriented Paradigm

OOP allows decomposition of a problem into a number of entities called objects and then builds data and function around these objects. The organization of data and function in object-oriented programs is shown in fig.1.3. The data of an object can be accessed only by the function associated with that object. However, function of one object can access the function of other objects.

Organization of data and function in OOP



EXAMPLE:
OBJECTS: STUDENT

DATA

Name
Date-of-birth
Marks

FUNCTIONS

Total
Average
Display
.....

Ques)Difference between procedure oriented and object oriented programming?
Some of the features of object oriented programming are;(3 mark ques)

- Importance is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external function.
- Objects may communicate with each other through function.
- New data and functions can be easily added whenever necessary.
- Follows bottom up approach in program design.
- Object-oriented programming is the most recent concept among programming paradigms and still means different things to different people.

Basic Concepts of OOP or Principles of OOP (Essay question)

It is necessary to understand some of the concepts used in object-oriented programming. These include:

1. Objects
2. Classes
3. Data abstraction and encapsulation
4. Inheritance
5. Polymorphism
6. Dynamic binding
7. Message passing

1.Objects

Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

When a program is executed, the objects interact by sending messages to one another.

fig 1.5 shows two notations that are popularly used in object- oriented analysis and design.

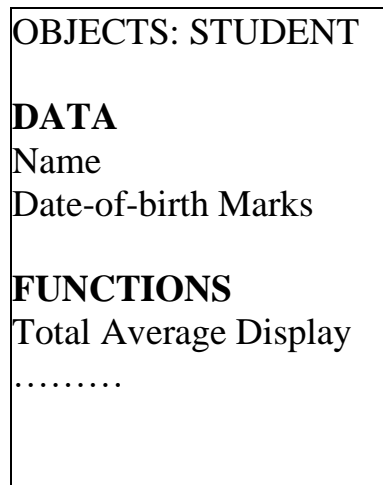


Fig. representing an object

2.Classes

The entire set of data and code of an object can be made a user-defined data type with the help of class. In fact, objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects of similar types.

For examples, Mango, Apple and orange members of class fruit. Classes are user-defined that types and behave like the built-in types of a programming language.

Syntax of creating a class,

Class classname;

Syntax of creating an object,

Classname objectname;

Example:

Class	Objects
fruit	Banana
	Apple
	Mango

If fruit has been defines as a class,

Class Fruit; //class name is Fruit
Fruit Mango; //Mango is the object of class Fruit

3.Data Abstraction

Abstraction refers to the act of representing essential features without including the background details or explanation. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, wait, and cost, and function operate on these attributes. They encapsulate all the essential properties of the object that are to be created.

The attributes are some time called ***data members*** because they hold information. The functions that operate on these data are sometimes called ***methods or member function***.

4.Encapsulation

The wrapping up of data and function into a single unit (called class) is known as ***encapsulation***. Data and encapsulation is the most striking feature of a class.

The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called ***data hiding or information hiding***.

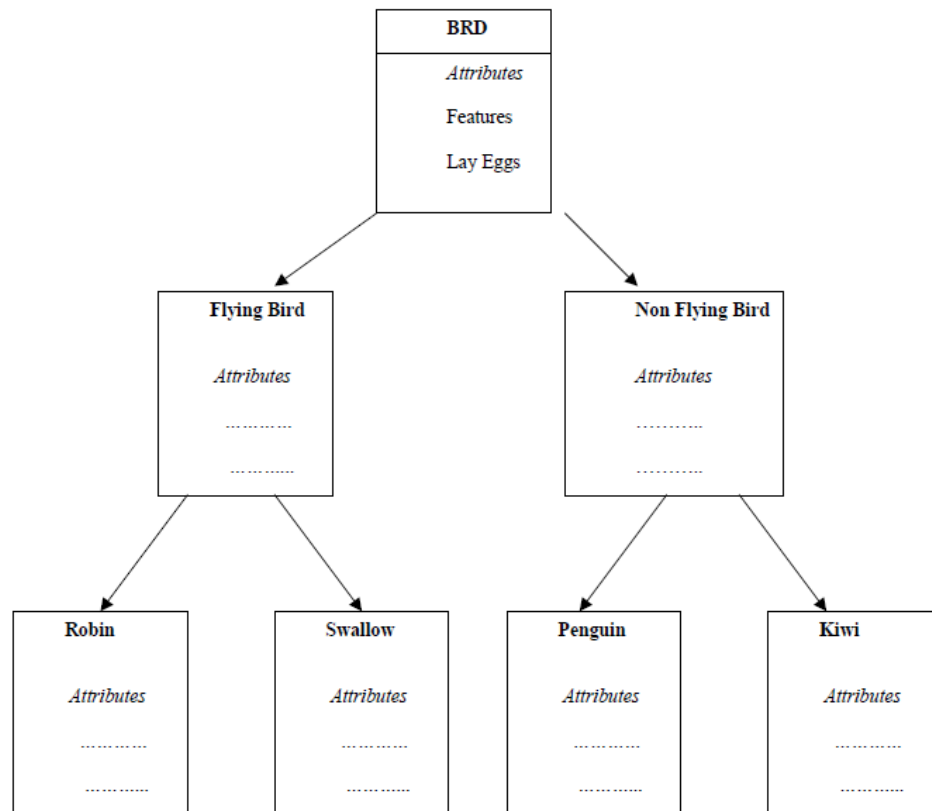
5.Inheritance

Inheritance is the process by which objects of one class acquired the properties of objects of another classes. It supports the concept of ***hierarchical classification***.

For example, the bird, 'robin' is a part of class 'flying bird' which is again a part of the class 'bird'. The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in fig 1.6.

In OOP, the concept of inheritance provides the idea of ***reusability***. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined feature of both the classes.

Fig. 1.6 Property inheritances



Advantage of inheritance: Allows the programmer to reuse a class i.e almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduced any undesirable side-effects into the rest of classes.

6.Polymorphism

Polymorphism is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than on form. An operation may exhibit different behavior is different instances. The behavior depends upon the types of data used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. The process of making an operator to exhibit different behaviors in different instances is known as *operator overloading*.

Fig. 1.7 illustrates that a single function name can be used to handle different number and different types of argument. This is something similar to a particular word having several different meanings depending upon the context. Using a single function name to perform different type of task is known as *function overloading*.

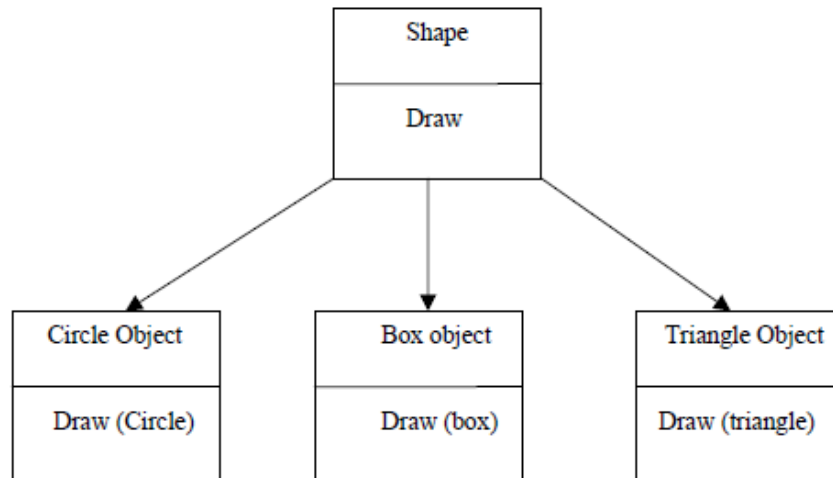


Fig. 1.7 Polymorphism

Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific action associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

7.Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. It is associated with polymorphism and inheritance. A function call associated with a polymorphic reference depends on the dynamic type of that reference.

8.Message Passing

An object-oriented program consists of a set of objects that communicate with each other. The process of programming in an object-oriented language, involves the following basic steps:
Creating classes that define object and their behavior,
Creating objects from class definitions, and
Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. The concept of message passing makes it easier to talk about building systems that directly model or simulate their real- world counterparts.

A Message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired results. *Message passing* involves specifying the name of object, the name of the function (message) and the information to be sent. Example:

Benefits of OOP

OOP offers several benefits to both the program designer and the user. The principal advantages are:

- Through inheritance, we can eliminate redundant code extend the use of existing Classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure program that can not be invaded by code in other parts of a programs.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map object in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more detail of a model can implemental form.
- Object-oriented system can be easily upgraded from small to large system.
- Message passing techniques for communication between objects makes to interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

Applications of OOP

OOP has become one of the programming buzzwords today. There appears to be a great deal of excitement and interest among software engineers in using OOP. Applications of OOP are beginning to gain importance in many areas.

The promising areas of application of OOP include:

- Real-time system
- Simulation and modeling
- Object-oriented data bases
- Hypertext, Hypermedia, and expert Ext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

Simple C++ Program

C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's.

Let us begin with a simple example of a C++ program that prints a string on the screen.

Printing A String

```
#include<iostream>
Using namespace std;
int main()
{
    cout<<" c++ is better than c \n";
    return 0;
}
```

Program 1

This simple program demonstrates several C++ features.

Program features

Like C, the C++ program is a collection of function. The above example contain only one function **main()**. As usual execution begins at main(). Every C++ program must have a **main()**. C++ is a free form language. Like C, the C++ statements terminate with semicolons.

Comments

C++ introduces a new comment symbol // (double slash). Comment start with a double slash symbol and terminate at the end of the line

The double slash comment is basically a single line comment. Multiline comments can be written as follows:

```
// This is an example of
// C++ program to illustrate
// some of its features
```

The C comment symbols `/*,*/` are still valid and are more suitable for multiline comments. The following comment is allowed:

```
/* This is an example of C++ program to illustrate some of its features
*/
```

Output operator

The statement

```
Cout<<"C++ is better than C.";
```

Causes the string in quotation marks to be displayed on the screen. This statement introduces two new C++ features, `cout` and `<<`. The identifier `cout` (pronounced as C out) is a predefined object that represents the standard output stream in C++. Here, the standard output stream represents the screen. It is also possible to redirect the output to other output devices. The operator `<<` is called the insertion or put to operator.

Input Operator

The statement

```
cin >> number1;
```

Is an input statement and causes the program to wait for the user to type in a number. The number keyed in is placed in the variable `number1`. The identifier `cin` (pronounced 'C in') is a predefined object in C++ that corresponds to the standard input stream. Here, this stream represents the keyboard.

The operator `>>` is known as extraction or get from operator. It extracts (or takes) the value from the keyboard and assigns it to the variable on its right

The iostream File

We have used the following `#include` directive in the program:

```
#include <iostream>
```

The `#include` directive instructs the compiler to include the contents of the file enclosed within angular brackets into the source file. The header file **`iostream.h`** should be included at the beginning of all programs that use input/output statements.

Namespace

Namespace is a new concept introduced by the ANSI C++ standards committee. This defines a scope for the identifiers that are used in a program. For using the identifier defined in the **namespace** scope we must include the using directive, like

Using namespace std;

Here, std is the namespace where ANSI C++ standard class libraries are defined. All ANSI C++ programs must include this directive. This will bring all the identifiers defined in std to the current global scope. **Using** and **namespace** are the new keyword of C++.

Variables

The program uses four variables number1, number2, sum and average. They are declared as type float by the statement.

```
float number1, number2, sum, average;
```

All variable must be declared before they are used in the program.

Return Type of main()

In C++, main () returns an integer value to the operating system. Therefore, every main () in C++ should end with a return (0) statement; otherwise a warning an error might occur. Since main () returns an integer type for main () is explicitly specified as **int**. Note that the default return type for all function in C++ is **int**. The following main without type and return will run with a warning:

```
main ()
{
    .....
    .....
}
```

More C++ Statements

AVERAGE OF TWO NUMBERS

```
#include<iostream.h> // include header file

Using namespace std;

Int main()

{

    Float number1, number2,sum, average;
    Cin >> number1; // Read Numbers Cin
    >> number2; // from keyboard Sum =
    number1 + number2;
    Average = sum/2;
    Cout << "Sum = " << sum << "\n";
    Cout << "Average = " << average << "\n";

    Return 0;

} //end of example
```

The output would be:

Enter two numbers: 6.5 7.5

Sum = 14

Average = 7

Cascading of I/O Operators

We have used the insertion operator << repeatedly in the last two statements for printing results.

The statement

```
Cout << "Sum = " << sum << "\n";
```

First sends the string "Sum = " to cout and then sends the value of sum.

Finally, it sends the newline character so that the next output will be in the new line. The multiple use of

<< in one statement is called cascading. When cascading an output operator,

we should ensure necessary blank spaces between different items. Using the cascading technique, the last two statements can be combined as follows:

```
Cout << "Sum = " << sum << "\n"  
<< "Average = " << average << "\n";
```

This is one statement but provides two line of output. If you want only one line of output, the statement will be:

```
Cout << "Sum = " << sum << ","  
<< "Average = " << average << "\n";
```

The output will be:

Sum = 14, average = 7

We can also cascade input iperator >> as shown below:

```
Cin >> number1 >> number2;
```

The values are assigned from left to right. That is, if we key in two values, say, 10 and 20, then 10 will be assigned to munber1 and 20 to number2

C++ Tokens

IDENTIFIERS: Identifiers are the names given to various program elements such as variables, functions and arrays. These are user defined names consisting of sequence of letters and digits.

Rules for declaring identifiers:

The first character must be an alphabet or underscore.

It must consist of only letters, digits and underscore.

Identifiers may have any length but only first 31 characters are significant.

It must not contain white space or blank space.

We should not use keywords as identifiers.

Upper and lower case letters are different.

Example: ab Ab aB AB are treated differently

Examples of valid identifiers:

a, x, n, num, SUM, fact, grand_total, sum_of_digits, sum1

Examples of Invalid identifiers: \$amount, ³num', grand-total, sum of digits, 4num.

\$amount : Special character is not permitted grand-total : hyphen is not

permitted.

sum of digits : blank spaces between the words are not allowed.

4num : should not start with a number (first character must be a letter or underscore

Note: Some compilers of C recognize only the first 8 characters only; because of this they are unable to distinguish identifiers with the words of length more than eight characters.

Variables: A named memory location is called variable.

OR

It is an identifier used to store the value of particular data type in the memory.

Since variable name is identifier we use following rules which are same as of identifier.

Rules for declaring Variables names:

The first character must be an alphabet or underscore.

It must consist of only letters, digits and underscore.

Identifiers may have any length but only first 31 characters are significant.

It must not contain white space or blank space.

We should not use keywords as identifiers.

Upper and lower case letters are different.

Variable names must be unique in the given scope

Ex: int a,b,a;//is invalid

int a,b;//is valid

Variable declaration: The declaration of variable gives the name for memory location and its size and specifies the range of value that can be stored in that location.

Syntax:

Data type variable name;

Ex:

int a=10;

float x=2.3;

KEYWORDS :

There are certain words, called keywords (reserved words) that have a predefined meaning in „C++“ language. These keywords are only to be used

for their intended purpose and not as identifiers.

The following table shows the standard „C++“ keywords

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while	class	friend	new	delete
this	public	private	protected	inline	try
throw	catch	template			

CONSTANTS:

Constants refer to values that do not change during the execution of a program.

Constants can be divided into two major categories:

Primary constants:

Numeric constants

Integer constants.

Floating-point (real)

constants. b) Character constants

Single character constants

String constants

Secondary constants:

Enumeration constants.

Symbolic constants.

Arrays, unions, etc.

Rules for declaring constants:

1.Commas and blank spaces are not permitted within the constant. 2.The constant can be preceded by minus (-) signed if required.

3.The value of a constant must be within its minimum bounds of its specified data type. **Integer constants:** An integer constant is an integer-valued number. It consists of sequence of digits. Integer constants can be written in three different number systems:

Floating point constants or Real constants : The numbers with fractional parts are called real constants.

Character constants:-

Single character constants: It is character(or any symbol or digit) enclosed

within single quotes. Ex: „a“ „1“ „*“.

Escape sequence

escape sequence are used in output functions. Some escape sequences are given below:

Escape sequence	Character
'\a'	audible alert
'\b'	back space
'\f'	form feed
'\n'	new line
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
'\"'	double quote
'\?'	question mark
'\\'	Backslash
'\0'	Null

OPERATORS AND EXPRESSIONS

An ***operator*** is a symbol which represents a particular operation that can be performed on data. An ***operand*** is the object on which an operation is performed.

By combining the operators and operands we form an ***expression***. An ***expression*** is a sequence of operands and operators that reduces to a single value.

operators can be classified as

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment or Decrement operators
- Conditional operator

ARITHMETIC OPERATORS :

operator	meaning
+	add
-	subtract
*	multiplication
/	division
%	modulo division(remainder)

An arithmetic operation involving only real operands(or integer operands) is called real arithmetic(or integer arithmetic). If a combination of arithmetic and real is called mixed mode arithmetic.

/*C program on Integer Arithmetic Expressions*/

```
#include<iostream.h> void main()
{
int a, b;
cout<<"Enter any two integers"; cin>>a>>b;
cout<<"a+b"<< a+b;
cout<<"a-b"<< a-b;
  cout<<"a*b"<< a*b;
  cout<<"a/b"<< a/b;
cout<<"a%b"<< a%b;
}
```

OUTPUT:

a+b=23 a-b=17 a*b=60 a/b=6 a% b=2

RELATIONAL OPERATORS :

We often compare two quantities and depending on their relation take certain decisions for that comparison we use relational operators.

operator	meaning
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equal to
!=	is not equal to

/* Eg: program to find largest of 2 nos*/(refer lab record)

LOGICAL OPERATORS:

In C++ we use int data type to represent logical data. If the data value is zero, it is considered as false. If it is non-zero (1 or any integer other than 0) it is considered as true. C++ has three logical operators for combining logical values and creating new logical values:

Logical AND &&

Logical OR ||

Eg: program to find largest of 3 nos.(Refer lab record)

ASSIGNMENT OPERATOR:

The assignment expression evaluates the operand on the right side of the operator (=) and places its value in the variable on the left.

Note: The left operand in an assignment expression must be a single variable. There are two forms of assignment:

Simple assignment

Compound assignment

Simple assignment :

In algebraic expressions we found these expressions. Ex: a=5; a=a+1; a=b+1;

Here, the left side operand must be a variable but not a constant. The left side variable must be able to receive a value of the expression. If the left operand cannot receive a value and we assign one to it, we get a compile error.

Compound Assignment:

A compound assignment is a shorthand notation for a simple assignment. It requires that the left operand be repeated as a part of the right expression.

Syntax: variable operator+=value

Ex:

A+=1; it is equivalent to A=A+1;

Advantages of using shorthand assignment operator:

What appears on the left-hand side need not be repeated and therefore it becomes easier to write.

The statement is more concise and easier to read.

The statement is more efficient.

Some of the commonly used shorthand assignment operators are shown in the following table:

Statement with simple assignment operator	Statement with shorthand operator
a=a+1	a+=1
a=a-1	a-=1
a=a*1	a*=1
a=a/1	a/=1
a=a%1	a%=1
a=a*(n+1)	a*=n+1

INCREMENT (++) AND DECREMENT (--) OPERATORS:

The operator ++ adds one to its operand where as the operator - - subtracts one from its operand. These operators are unary operators and take the following form:

Operator	Description
++a	Pre-increment
a++	Post-increment
--a	Pre-decrement
a--	Post-decrement

Both the increment and decrement operators may either precede or follow the operand.

Postfix Increment/Decrement :(a++/a--)

In postfix increment (Decrement) the value is incremented (decremented) by one. Thus the

a++ has same effect as a=a+1; a--has the same effect as a=a-1.

The difference between a++ and a+1 is, if ++ is after the operand, the increment takes place after the expression is evaluated.

The operand in a postfix expression must be a variable.

Ex1:

int a=5;

B=a++; Here the value of B is 5. the value of a is 6.

Ex2:

int x=4; y=x--; Here the value of y is 4, x value is 3

Prefix Increment/Decrement (++a/ --a)

In prefix increment (decrement) the effect takes place before the expression that contains the operator is evaluated. It is the reverse of the postfix operation. ++a has the same effect as a=a+1.

--a has the same effect as a=a-1.

Ex: int b=4;

A= ++b;

In this case the value of b would be 5 and A would be 5.

The effects of both postfix and prefix increment is the same: the variable is incremented by

But they behave differently when they used in expressions as shown above.

The execution of these operators is fast when compared to the equivalent assignment statement.

Example:

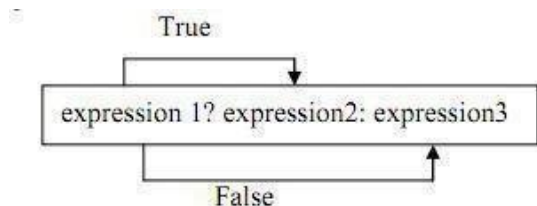
```
int main()
{
int a=1; int b=5;
++a;
cout<<"a="<<a;
--b;
cout<<"b="<<b;
cout<<"a="<<a++;
cout<<"a="<<a;
cout<<"b="<<b--;
cout<<"b="<<b;
return 0;
}
```

output

a=2 b=4 a=2 a=3 b=4 b=3

CONDITIONAL OPERATOR OR TERNARY OPERATOR:

A ternary operator requires two operands to operate Syntax:



```
#include<iostream.h> void main()
{
int a, b,c;
cout<<"Enter a and b values:"; cin>>a>>b;
c=a>b?a:b;
cout<<"largest of a and b is "<<c;
}
```

Output:

Enter a and b values:1 5
largest of a and b is 5

Other Operators in c++: All above operators of c language are also valid in c++.New operators introduced in c++ are

Sno	Operator	Symbol
1.	Scope resolution operator	::
2.	Pointer to a member declarator	::*
3.	Pointer to member operator	->*, ->
4.	Pointer to member operator	.*
5.	new	Memory allocating operator
6.	delete	Memory release operator
7.	endl	Line feed operator
8.	setw	Field width operator
9.	insertion	<<
10.	Extraction	>>

Loop control statements or repetitions:

A block or group of statements executed repeatedly until some condition is satisfied is called Loop.

The group of statements enclosed within curly brace is called block or compound statement.

We have two types of looping structures.**Entry control and exit control**

One in which condition is tested before entering the statement block called **entry control**.

The other in which condition is checked at exit called **exit controlled loop**.

Loop statements can be divided into three categories as given below

- 1.while loop statement
- 2.do while loop statement
- 3.for loop statement

1.WHILE LOOP:

While(test condition)

{

body of the loop

```
}
```

It is an entry controlled loop. The condition is evaluated and if it is true then body of loop is executed. After execution of body the condition is once again evaluated and if it is true body is executed once again. This goes on until test condition becomes false.

/* program to find sum of n natural numbers */

```
#include<iostream>
```

```
int main()
```

```
{
```

```
int i = 1,sum = 0,n;
```

```
cout<<"Enter N"<<end; cin>>n;
```

```
while(i<=n)
```

```
{
```

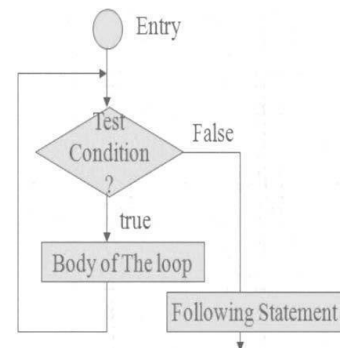
```
sum = sum + i; i = i + 1;
```

```
}
```

```
cout<<"Sum of first"<<n<<"natural numbers is:"<<sum;
```

```
return 0;
```

```
}
```



2.DO WHILE LOOP:

The while loop does not allow body to be executed if test condition is false.

The do while is an exit controlled loop and its body is executed at least once.

```
do
```

```
{
```

```
body
```

```
}while(test condition);
```

/* program to find sum of n natural numbers */

```
int main()
```

```
{
```

```
int i = 1,sum = 0,n;
```

```
cout<<"Enter N";
```

```
cin>>n
```

```
do{
```

```
sum = sum + i;
```

```
i = i + 1;
```

```
} while(i<=n);
```

```
cout<<"Sum of first"<< n<<" natural numbers is:"<<sum; return 0;
}
```

Note: if test condition is false. before the loop is being executed then While loop executes **zero** number of times where as do--while executes **one** time

3.FOR LOOP :It is also an entry control loop that provides a more concise structure

Syntax:

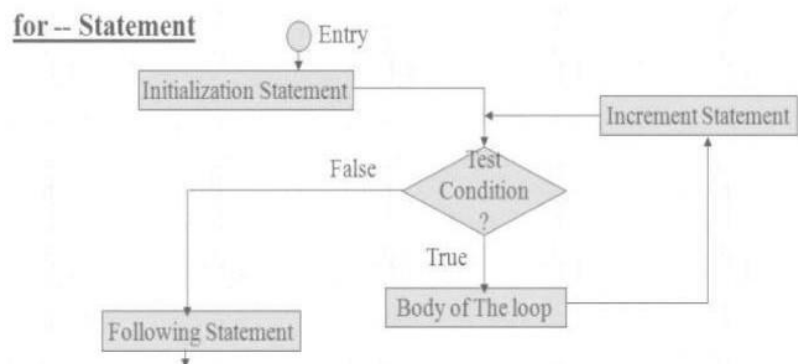
```
for(initialization; test expression; increment/decrement)
{
statements;
}
```

For statement is divided into three expressions each is separated by semi colon;

initilization expression is used to initialize variables

2.test expression is responsible of continuing the loop. If it is true, then the program control flow goes inside the loop and executes the block of statements associated with it .If test expression is false loop terminates

3.increment/decrement expression consists of increment or decrement operator This process continues until test condition satisfies.



Example of for loop:

/* program to find sum of n natural numbers */

#include<stdio.h>

int main()

{

int i ,sum = 0,n;

cout<<"Enter N"; cin>>n;

for(i=1;i<=n;i++)

{

sum = sum + i;

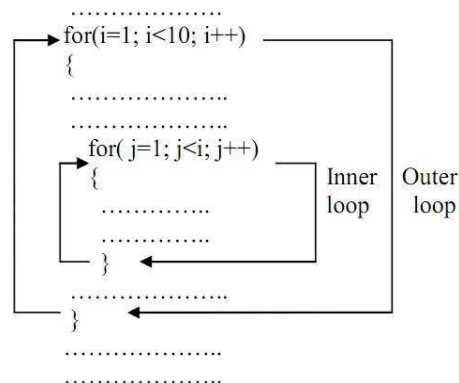
}

cout<<"Sum of first"<<n<<" natural numbers is:%d"<<sum; return 0;

}

Nested loops:

Writing one loop control statement within another loop control statement is called nested loop statement



Ex:

for(i=1;i<=10;i++)

for(j=1;j<=10;j++)

cout<<i<<j;

/*program to print prime numbers upto a given number*/

void main()

{

int n,i,fact,j; clrscr();

cout<<"enter the number:"; cin>>n

for(i=1;i<=n;i++)


```

{fact=0;
//THIS LOOP WILL CHECK A NO TO BE PRIME NO. OR
NOT. for(j=1;j<=i;j++)
{
if(i%j==0) fact++;
}
if(fact==2)
cout<<i<<"\t";
}
getch( );
}

```

Output:

Enter the number : 5
2 3 5

Unconditional control statements:

Statements that transfers control from on part of the program to another part unconditionally Different unconditional statements are

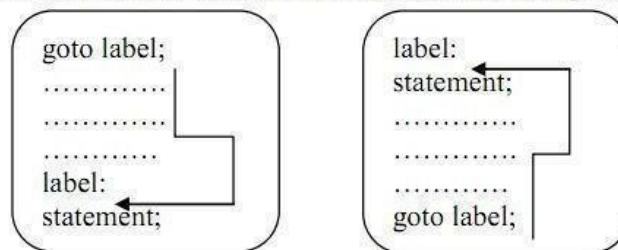
goto

break

continue

1. goto :- **goto statement** is used for unconditional branching or transfer of the program execution to the labeled statement.

The goto statement to branch unconditionally from one point to another in the program. The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be followed by colon (;). The label is placed immediately before the statement where the control is to be transferred. The general form of goto is shown below:



Forward Jump

Backward Jump

The label: can be anywhere in the program either before or after the goto label; statement.

If the label: is placed after the goto label;, some statements will be skipped and jump is known as a Forward Jump.

If the label: is placed before the goto label; a loop will be formed some statements will be executed repeatedly. Such a jump is known as a Backward Jump.

```

/* program to find sum of n natural numbers */
#include<stdio.h>
int main()
{
int i ,sum = 0,n;
cout<<"Enter N"; cin>>n;
i=1; L1:
sum = sum + i;
i++;
if(i<=n) goto L1;
cout<<"Sum of first "<<n<<" natural numbers is"<<sum; return 0;
}

```

2.break:-

when a break statement is encountered within a loop ,loop is immediately exited and the program continues with the statements immediately following loop

```

/* program to find sum of n natural numbers */
int main()
{
int i ,sum = 0,n;
cout<<"Enter N"; cin>>n;
i=1;
L1:
sum = sum + i;
i++;
if(i>n) break;
goto L1;
cout<<"Sum of first"<<n<<"natural numbers is: "<<sum; return 0;
}

```

3.Continue:It is used to continue the iteration of the loop statement by skipping the statements after continue statement. It causes the control to go directly to the test condition and then to continue the loop.

```

/* program to find sum of n positive numbers read from keyboard*/
int main()
{
int i ,sum = 0,n,number;
cout<<"Enter N";

```

```

    cin>>n;
    for(i=1;i<=n;i++)
    {
        cout<<"Enter a number:";
        cin>>number;
        if(number<0)
            continue;
        sum = sum + number;
    }
    cout<<"Sum of"<<n<<" numbers is:"<<sum;
    return 0;
}

```

Classes in C++

An object oriented programming approach is a collection of objects and each object consists of corresponding data structures and procedures. The program is reusable and more maintainable. The important aspect in OOP is a class which has similar syntax that of structure.

class: It is a collection of data and member functions that manipulate data. The data components of class are called data members and functions that manipulate the data are called member functions. Class is declared by the keyword class.

Syntax:-

```

class classname
{
    Access specifier :
    Variable declarations; //data member
    Access specifier :
    function declarations; //member function
};

```

Example:

```

class Student
{
    public:
    int roll;
    char name[20];
    public:
    void getdata();
    void display();
}

```

Access specifier or access modifiers or Access Control OR Visibility modes

Access specifier or access modifiers are the labels that specify type of access given to members of a class. These are used for data hiding. These are also called as visibility modes.

There are three types of access specifiers

- 1.private
- 2.public
- 3.protected

1.Private:

1. If the data members are declared as private access then they cannot be accessed from other functions outside the class.
2. It can only be accessed by the functions declared within the class.
3. It is declared by the keyword **private**.

2.public:

1. If the data members are declared public access then they can be accessed from other functions out side the class.
2. It is declared by the key word **public** .

3.protected:

1. The access level of protected declaration lies between public and private.
2. This access specifier is used at the time of inheritance.

Note:-

- ▶ If no access specifier is specified then it is treated by default as private
- ▶ The default access specifier of structure is public where as that of a class is “private”

▶ Example:

```
class Student
{
private :
int roll;
char name[30];
public:
void get_data()
{
cout<<"Enter roll number and name":
cin>>roll>>name;
}
void put_data()
```

```
{
cout<<"Roll number:"<<roll<<endl;
cout<<"Name :"<<name<<endl;
}
};
```

Declaring Objects

Object:-Instance of a class is called object.

Syntax:

class_name object_name;

Ex:

student s; //s is an object of class student

Accessing members:-dot operator is used to access members of class

Object-name.function-name(actual arguments);

Example

```
int main()
{
Student s;
s.getdata(); //dot operator is used for accessing member function
s.putdata();
return 0;
}
```

FUNCTION DEFINITION AND DECLARATION

☐ Functions are the building blocks of C++ programs where all the program activity occurs.

☐ Function is a collection of declarations and statements.

☐ In C++, a function must be defined prior to its use in the program. The function definition contains the code for the function.

The general syntax of a function definition in C++ is shown below:

Return type name_of_the_function (argument list)

```
{
//body of the function
}
```

☐ The type specifies the type of the value to be returned by the function. It may be any valid C++ data type.

☐ When no type is given, then the compiler returns an integer value from the function.

☐ name_of_the_function is a valid C++ identifier (no reserved word allowed) defined by the user.

- ❑ Argument list is a comma separated list of variables of a function through which the function may receive data or send data when called from other function.
- ❑ When no parameters, the argument list is empty.

C++ User-defined Function Types

1. Function with no argument and no return value.
2. Function with no argument but return value.
3. Function with argument but no return value.
4. Function with argument and return value

1. Function with no argument and no return value.

<pre>#include <iostream> using namespace std; class Demo { public: int a,b; void get() { cout<<"enter 2 nos"; cin>>a>>b; } void sum() { cout<<"Sum="<<a+b; } };</pre>	<pre>int main() { Demo obj; //object created obj.get(); // function call obj.sum(); return 0; } Output Enter 2 nos 2 3 Sum=5</pre>
---	---

2.Function with no argument but return value.

<pre>#include <iostream> using namespace std; class Demo { public: int a,b,s; void get() { cout<<"enter 2 nos"; cin>>a>>b; } int sum() { s=a+b; return s; } };</pre>	<pre>int main() { Demo obj; //object created obj.get(); // function call int add=obj.sum(); Cout<<"Sum="<<add; return 0; } Output Enter 2 nos 2 3 Sum=5</pre>
--	---

3.Function with argument but no return value.

<pre>#include <iostream> using namespace std; class Demo { public: void sum(int a,int b) { cout<<"Sum="<<a+b; } };</pre>	<pre>int main() { Demo obj; //object created int x,y; Cout<<"enter 2 no.s"; Cin>>x>>y; obj.sum(x,y); return 0; } Output Enter 2 nos 2 3 Sum=5</pre>
--	---

4.Function with argument and return value.

<pre>#include <iostream> using namespace std; class Demo { public: int a,b,s; int sum(int a,int b) { s=a+b; return s; } };</pre>	<pre>int main() { Demo obj; int x,y; cout<<"enter 2 no.s"; cin>>x>>y; int add=obj.sum(x,y); cout<<"sum="<<add; return 0; }</pre> <p>Output Enter 2 nos 2 3 Sum=5</p>
--	--

Defining Member Function[Essay Questions]

► Member function Can be defined in two places:

1. Outside the class definition
2. Inside the class definition

Outside the class definition

► An important difference between a member function and a normal function is that a member function incorporates a membership “identity label” in the header.

► This label tells the compiler which class the function belongs to.

► The general form of a member function definition is:

```
returntype class_name :: function_name(argument declaration )
{
    Function body;
}
```

► The membership label `class_name ::` tells the compiler that the function ***function_name*** belongs to the ***class class_name***.

► That is the scope of the function is restricted to the class name specified in the header line.

► The symbol :: is called the scope resolution operator.

Example:

```
void Item::getdata(int a,int b)
{
    number=a;
    cost=b;
}
```

Example program:

```
class Demo{
public:
    int age;
    string name;
    void getdata();
    void disp();
};
void Demo::getdata()
{
    cout<<"enter age and name";
    cin>>age>>name;
}
void Demo::disp()
{
    cout<<"Age="<<age<<"\n";
    cout<<"name="<<name;
}
```

```
int main()
{
    Demo obj;
    obj.getdata();
    obj.disp();
    return 0;
}
```

2. Inside class definition

Another method of defining a member function is to replace the function declaration by the function definition inside the class.

► Example program:

<pre>#include <iostream> using namespace std; class Demo { public: int a,b; void get() { cout<<"enter 2 nos"; cin>>a>>b; } void sum() { cout<<"Sum="<<a+b; } };</pre>	<pre>int main() { Demo obj; //object created obj.get(); // function call obj.sum(); return 0; } Output Enter 2 nos 2 3 Sum=5</pre>
---	---

Characteristics of Member Functions

The member function have some special characteristics, they are:

1. Several different classes can use the same function name. The membership label will resolve the scope.
2. Member function can access the private data of the class. A non-member function cannot do so.
3. A member function can call another member function directly, without using the dot operator.

Array with in a class

An array is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They can be used to store collection of primitive data types such as int, float, double, char, etc of any particular type.

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

<- Array Indices

Array Length = 9

First Index = 0

Last Index = 8

- ▶ Array can be used as member variables in a class.
- ▶ Arrays can be declared as the members of a class.
- ▶ The arrays can be declared as private, public or protected members of the class.

To understand the concept of arrays as members of a class, consider this example....

```
#include<iostream>
class Student
{
int roll_no;
int marks[10];
public:
void getdata ();
void tot_marks ();
};
void Student :: getdata ()
{
cout<<"\nEnter roll no: ";
cin>>roll_no;
for(int i=1; i<=5; i++)
{
cout<<"Enter marks in subject"<<i<<": ";
```

```

cin>>marks[i] ;
}
}
void Student :: tot_marks() //calculating total marks
{
    int total=0;
    for(int i=1; i<=5; i++)
    {
        total=total+marks[i];
    }
    cout<<"\n\nTotal marks "<<total;
}
int main()
{
    Student obj;
    obj.getdata() ;
    obj.tot_marks() ;
    return 0;
}

```

Output:

```

Enter roll no: 101
Enter marks in subject 1: 67
Enter marks in subject 2 : 54
Enter marks in subject 3 : 68
Enter marks in subject 4 : 72
Enter marks in subject 5 : 82
Total marks = 343

```

Example2:

```

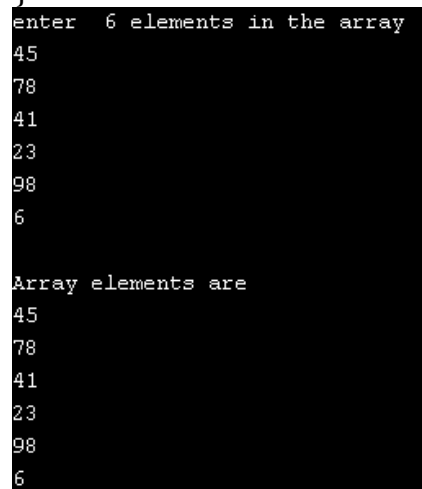
#include<iostream>
using namespace std;
class Arraydemo
{
    int a[6];
    public:
    void getelts();
    void show();
}

```

```

};
void Arraydemo::getelts()
{
    cout<<"enter 6 elements in the array";
    for(int i=1;i<=6;i++)
    {
        cin>>a[i];
    }
}
void Arraydemo::show()
{
    cout<<"\nArray elements are"<<endl;
    for(int i=1;i<=6;i++)
    {
        cout<<a[i]<<endl;
    }
}
int main()
{
    Arraydemo obj;
    obj.getelts();
    obj.show();
    return 0;
}

```



```

enter 6 elements in the array
45
78
41
23
98
6

Array elements are
45
78
41
23
98
6

```

Array of objects

- Like array of other user-defined data types, an array of type class can also be created.
- The array of type class contains the objects of the class as its individual elements.

- Thus, an array of a class type is also known as an array of objects.
- An array of objects is declared in the same way as an array of any built-in data type.

```

class Employee
{
char name[30];
float age;
Public:
void getdata();
void putdata();
};
Void Employee::getdata()
{
Cout<<"enter name";
Cin>>name;
Cout<<"enter age";
Cin>>age;
}
void Employee::putdata()
{
Cout<<"name"<<name;
Cout<<"age"<<age;
}
int main()
{
Employee obj1[5]; //array of object1
For(i=1;i<=5;i++)
{
obj1[i].getdata();
}
For(i=1;i<=5;i++)
{
Obj1[i].putdata();
}
}

```

Examples:

Employee obj1[3]; //array of object1

Employee obj2[4]; //array of object2

Employee obj3[2]; //array of object3

- ▶ In the above eg. Obj1 contains three objects, namely obj1[0], obj1[1], obj1[2]
- ▶ Obj2 contains three objects, namely obj2[0], obj2[1], obj2[2], obj2[3]
- ▶ Obj3 contains three objects, namely obj3[0], obj3[1]
- ▶ We can use the usual array accessing methods to access individual elements, and then the dot operator to access the member functions.
- ▶ Eg: **obj1[i].getdata();**
- ▶ Eg: **obj1[i].putdata();**
- ▶ **That is this statement request the object obj1[i] to invoke the member function getdata();**

Object as function argument

To pass an object as an argument we write the object name as the argument while calling the function the same way we do it for other variables.

Syntax:

function_name(object_name);

Example:

Student obj1,obj2; //obj1 and obj2 are 2 objects of class Student
obj1.add(obj2); //passing obj2 as an argument to function add;

Example

```
#include <iostream>
using namespace std;
class Demo
{
public:
int a;
public:
    void set(int x)
    {
        a = x;
    }
    void sum(Demo ob1, Demo ob2)
    {
        a= ob1.a + ob2.a;
    }
    void print()
    {
        cout<<"Value of A : "<<a<<endl;
    }
};

int main()
{
    //object declarations
    Demo d1;
    Demo d2;
    Demo d3;
    //assigning values to the data member of objects
    d1.set(10);
    d2.set(20);
    //passing object d1 and d2
    d3.sum(d1,d2);
    //printing the values
    d1.print();
    d2.print();
    d3.print();
    return 0;
}
```

Output

```
Value of A : 10
Value of A : 20
Value of A : 30
```